Practical 2: Implementation of lexical analyser using Lex tool.

Aim: (Tokenizing) Use Lex and yacc to extract tokens from a given source code.

Description:

- A language for specifying lexical analyzer.
- There is a wide range of tools for construction of lexical analyzer. The majority of these tools arebased on regular expressions.
- The one of the traditional tools of that kind is lex.

Lex:-

- The lex is used in the manner depicted. A specification of the lexical analyzer is preferred by creating a program lex.1 in the lex language.
- Then lex.1 is run through the lex compiler to produce a 'c' program lex.yy.c.
- The program lex.yy.c consists of a tabular representation of a transition diagram constructed fromthe regular expression of lex.1 together with a standard routine that uses table of recognize leximes.
- Lex.yy.c is run through the 'C' compiler to produce as object program a.out, which is the lexical
  analyzer that transforms as input stream into sequence of tokens.

Algorithm:

1. First, a specification of a lexical analyzer is prepared by creating a program lexp.l in the LEX language.
2. The Lexp.l program is run through the LEX compiler to produce an equivalent code inC language named Lex.yy.c
3. The program lex.yy.c consists of a table constructed from the Regular Expressions ofLexp.l, together with standard routines that uses the table to recognize lexemes.
4. Finally, lex.yy.c program is run through the C Compiler to produce an object programa.out, which is the lexical analyzer that transforms an input stream into a sequence of tokens.

Program

```
lexp.l
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf ("\n %s is a Preprocessor Directive",yytext);} int |
float | main | if | else | printf | scanf | for | char | getch |
while {printf("\n %s is a Keyword",yytext);} "/*" {COMMENT=1;}
"*/" {COMMENT=0;}
{identifier}\( {if(!COMMENT) printf("\n Function:\t %s",yytext);}
\{ {if(!COMMENT) printf("\n Block Begins");
\} {if(!COMMENT) printf("\n Block Ends");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s is an
Identifier",yytext);}
\".*\" {if(!COMMENT) printf("\n %s is a String",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a Number",yytext);}
\)(\;)? {if(!COMMENT) printf("\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT) printf("\n%s is an Assmt oprtr",yytext);}
\<= |
\>= |
\< |
== {if(!COMMENT) printf("\n %s is a Rel. Operator",yytext);}
.|\n
%%
int main(int argc, char **argv)
{
if(argc>1)
{
FILE *file; file=fopen(argv[1],"r"); if(!file)
{
printf("\n Could not open the file: %s",argv[1]); exit(0);
}
yyin=file;
}
yylex(); printf("\n\n"); return 0;
}
int yywrap()
{
return 0;
}
Output:
test.c #include<stdio.h> main()
{
int fact=1,n;
for(int i=1;i<=n;i++)
{ fact=fact*i; }
printf("Factorial Value of N is", fact); getch();
}

$ lex lexp.l
$ cc lex.yy.c
$ ./a.out test.c
#include<stdio.h> is a Preprocessor Directive Function: main( )
Block Begins
int is a Keyword fact is an Identifier
= is an Assignment Operator
1 is a Number
n is an Identifier Function: for( int is a Keyword i is an Identifier
= is an Assignment Operator 1 is a Number
i is an Identifier
<= is a Relational Operator n is an Identifier
i is an Identifier
);
Block Begins
fact is an Identifier
= is an Assignment Operator fact is an Identifier
i is an Identifier Block Ends Function: printf(
"Factorial Value of N is" is a String fact is an Identifier );
Function: getch( ); Block Ends
```